An Efficient Delivery of Historical Information for The Mendelian Inheritance in Man Database

Peter Li, David Waldo, Stuart Pineo, and Patricia Foster
Genome Database, Johns Hopkins University, Baltimore, MD

The ability to manage information with regard to changes in a database is critical for quality control. This information can also provide audit trails about the time of the change and the person who made the change. In addition, historical information can provide the proper context in which to interpret the relationships between the current and past data. In most genomic databases, only the most recent copy of the information is presented to the user, thereby losing the audit trail and the historical context. Therefore, we have constructed a delivery mechanism for the historical information in the Mendelian Inheritance in Man *database. Futhermore, this feature was designed to optionally display only the changes so that the user can bypass the unchanged portions of the text. It was anticipated that technical problems would influence the acceptance of this information delivery. However, the involvement of the editorial staff became the critical factor.*

INTRODUCTION

There are many uses for historical information in a database. One such use is for audit trails in quality control. For example, a quality examiner would like to know when a change was made in the database or who made the change, especially if the change has legal or financial implications. Another use for historical information is to represent consistent inter-data linking. For example, if several pieces of data were linked together based on the semantics of the data, then a change in one of the component data may alter the semantics and, consequently, may affect the links. This can lead to complex and cascaded link updates. One solution is to link with an historical state of that data, which will always remain semantically valid. A final use for historical information is to study the trends in a domain. For example, if the database is used to support a scientific endeavor, one can study the historical changes to the database entries as indicators for trends in that domain.

There are several methods of managing historical information [1]. One way is to describe explicitly the date in the schema and associate a date with each changing data value. This method has high cognitive cost because the user or the application for such a database must build the historical context using temporal logic. A second way is to use data versioning, a concept for managing historical or alternative data sets that are related to a single database object, not unlike versions of a journal article manuscript. Data versioning can be implicit, where old data are automatically archived when a change is applied. Recent commercial databases have incorporated such a concept [2], but the burden is on storage management, i.e., what portion of the data are versioned when only part has been changed. A third approach is to use explicit versioning, i.e., specific data are defined to be versioned, so that only these will be archived when changes occur. Another consideration is how the changes are stored. The previous methods refer to archiving old data by keeping a copy of that data in the database. However, it is just as valid to store the "change process," i.e., as a transform that can be applied to the data. The trade-off is the space needed to store the older copies vs. the time needed to compute the older data based on the stored change processes.

While all these methods apply to text and nontext databases alike, the advances in historical databases mainly occurred in database technology such as relational and object-oriented DBMS [3]. On the other hand, text databases are usually considered static, where every piece of text, once entered, is immutable. Certain textual databases, e.g., medical records, are "append-only" where new information is only added at the end of existing entries [4]. These can be adequately managed with the current technology of date-stamping. However, there is also a category of textual databases, such as Mendelian Inheritance in Man (MIM) [5], where changes include replacement and deletion of text. Other examples include the summary of a patient's medical history or the active problem list where changes occur in a "nonappend" manner. However, these examples are often encoded using relational means, thereby losing the free text search feature associated with text databases.

In this article, we use the MIM database as an example to explore the issues involved in the construction and delivery of historical information: the simplifying assumptions, the common UNIX utilities used, and the major design issues that affected the acceptance of this database extension.

TECHNICAL DESIGN

To support full historical access, one should be able to formulate queries for a version of the database at any given time in the past. Under this scenerio, all information (past and present) must be indexed for efficient retrieval. However, this is currently beyond the capability of our IRx text search engine [6]. Therefore, we focused on the delivery of audit trail information while developing a more capable search engine.

The MIM Database

Mendelian Inheritance in Man (MIM) is a comprehensive catalog of genes and inheritable human phenotypes. It is an online text database composed of 7,200 entries describing the clinical aspects of diseases and the molecular aspects of genes. Each entry is encoded in Standard Generalized Markup Language (SGML) [7]. The architecture of this database has been discussed elsewhere [8]. Briefly, MIM is built with UNIX directories and text files placed under Revision Control System (RCS) [9]. All accesses (retrievals and updates) are governed by several UNIX shell scripts and perl programs [10]. This SGML database is converted to many different distribution formats: printed form (book), CDROM for PC-based retrieval, flat ASCII files for Internet file transfer, and HTML-encoded files for World-Wide Web (WWW) access [11]. Because the WWW access (http://gdbwww.gdb.org/omim/docs/omimtop.html) is the most active distribution method, we chose to deliver the historical information via WWW.

Prior to SGML conversion, MIM entries were organized in "diachronic" fashion, i.e., new material was added to the end of the text with minimal changes to the previously entered text. With the SGML conversion, a new format was introduced into the database that supported "synchronic" organization of information, where the text is broken down into several topical sections with a section dedicated to historical information [5]. However, this does not ensure that changing an entry will automatically migrate the old text to that section. Therefore, an another mechanism was needed to provide historical changes in the proper context.

RCS History

Revision Control System (RCS) is a tool that archives different versions of a file. It stores the most recent version as-is, but only the differences needed to regenerate the previous versions. This is a common tool used in software development and in UNIX document management. The MIM database has been using RCS for the past 7 years. However, the conversion of entries to the SGML format forced a break in the archive history, and therefore, only post-SGML conversion changes, after January 1994, are available through current RCS archives.

When a MIM entry file is updated via the "checkin" command, information about the person and the time is added to the archive. It also asks for a brief summary of the changes that were made to the entry. This summary is stored as part of the revision log. RCS provides a "rlog" command to retrieve just the log of changes. The output of "rlog" is converted to a list of choices in an HTML (WWW hypertext) file format (Fig. 1). The end-user can then select the particular version to view through a WWW browser.

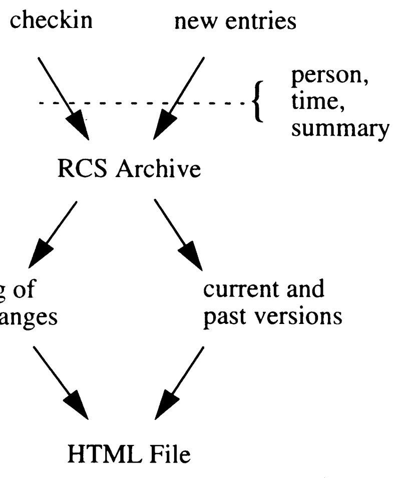

Figure 1: RCS to HTML

UNIX "diff"

With RCS, we can deliver any previous version of the entry. However, due to the editorial nature of MIM entries, it is just as important to show only the changes between one version against another. A UNIX utility, named "diff" [12], can determine the lines of text that were changed from one version to the next. For historical queries, "diff" is applied to show the changes between two previous versions of an entry and between any version against the current ver-

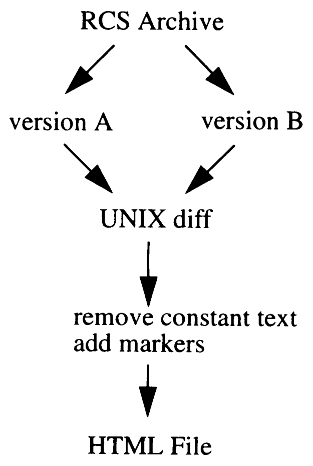

Figure 2: Generating only the changes

sion. The output of "diff" is converted to HTML (Fig. 2) by adding distinquishing markers to the boundaries of changed sections and removing the unchanged portion of the text between the versions. The use of "rlog" and "diff" with various HTML conversion programs gave us a simple and effective method to track and show changes to a MIM entry.

DISCUSSION

As we implemented the history delivery mechanism, we focused on three major technical design issues and one nontechnical issue.

Word vs. Line Granularity

UNIX "diff" computes the differences between files at the granularity of a line of text, terminated by the ASCII character NEWLINE. This is not sufficient granularity because insertion or deletion of a single phrase can alter the placement of words on many subsequent text lines. The end result is many more lines are marked as changed when, in fact, they have not. For example, if the starting text was formatted as follows (¶ represents NEWLINE):

Among the 15 molecular variants of AGT¶
that had been identified, significant¶
association with hypertension was observed¶
with 2 amino acid substitutions, M235T and¶
T174M. These 2 variants exhibited¶
complete linkage disequilibrium, as T174M¶
occurred on a subset of the haplotypes¶
carrying the M235T variant, and both¶
haplotypes were observed at higher¶
frequency among hypertensives.¶

Then if a MIM reference number "(106150.0001)" was added after M235T, the text will reformat to:

Among the 15 molecular variants of AGT¶
that had been identified, significant¶
association with hypertension was observed¶
with 2 amino acid substitutions, M235T¶
<u>(106150.0001)</u> and T174M. These 2¶
variants exhibited complete linkage¶
disequilibrium, as T174M occurred on a¶
subset of the haplotypes carrying the¶
M235T variant, and both haplotypes were¶
observed at higher frequency among¶
hypertensives.¶

Notice all the line breaks (¶) after M235T have been changed and will be reported as changed by "diff". Therefore, a "diff" at the word granularity level is needed. However, instead of writing a new "diff" program from scratch, the SGML text files were converted to one word per line, then the standard "diff" program could determine the differences at the word level. On the other hand, the side effect of this conversion was a 20% increase in the size of text files by the addition of the NEWLINE character after each word, based on an average word size of 5 characters. Two additional UNIX/perl programs are used to interconvert this word-per-line internal file format to the formats needed for distribution or editing (Fig. 3). This modest increase in both size and performance overhead is considered acceptable for providing a more meaningful set of changes to the end-user.

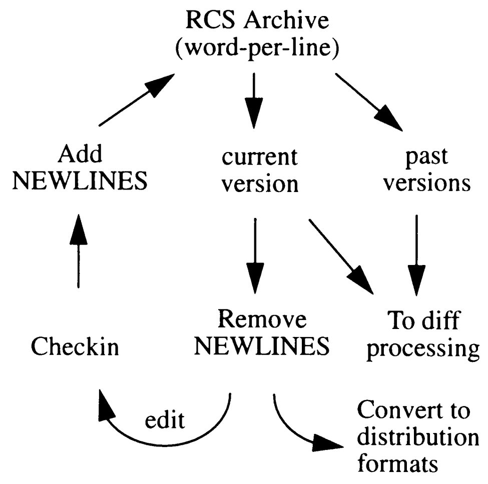

Figure 3: Interconversions to support different formats.

SGML Tags

The "diff" program does not understand the hierarchical organization of SGML-tagged text: the tags are just text lines to be processed. This results in differences that cross SGML tag boundaries. For example, if the starting paragraph is the following:

<P>*Alder (1939) originally described the anomaly in a brother and sister who later at puberty developed changes in their hip joints. The brother was said to be in good health at age 28 (Davidson, 1961). Jordans (1947) reported a Dutch family showing a dominant inheritance pattern--9 affected persons in 3 generations with male-to-male transmission.***</P>**

Where "<P>" and "</P>" are SGML tags. Then if we split the paragraph by adding new text (underlined) in the middle.

<P>*Alder (1939) originally described the anomaly in a brother and sister who later at puberty developed changes in their hip joints. The brother was said to be in good health at age 28 (Davidson, 1961).*

<ADDED>*This was, in fact, not true (Steinmann, 1994).*</P>
<P>*Alder (1939) described the granules in a 9-year-old girl with scarlet fever.*</ADDED> *Jordans (1947) reported a Dutch family showing a dominant inheritance pattern--9 affected persons in 3 generations with male-to-male transmission.*</P>

The resulting start tag of addition (**<ADDED>**) does not close when it reaches the end of the leading paragraph. Although SGML encoding has the option to close the **<ADDED>** tag automatically as a solution in the leading paragraph, it does not have an option to start a second **<ADDED>** tag in the trailing paragraph. To solve this problem, a postprocessing program reads the "diff" output and adds the correct tags to produce a file that remains a valid SGML document. An added benefit of this postprocessing is that the resultant files can be loaded into any SGML editor with the appropriate highlights for the changed text.

Schema Evolution

A characteristic of genomic databases is their rapid schema evolution as new technology and science reorganize the stored information. The MIM database is no exception to this evolution: its schema, the SGML Document Type Definition (DTD), has been changed several times to support new structures and to handle reorganizations. This rapid evolution presented a major design problem because a historical retrieval can retrieve a version of an entry that is compliant to an earlier DTD. Consequently, the HTML conversion program must now ensure that its output remains meaningful independent of current DTD structure. Fortunately, only the upper layer structures changed between revisions and, in turn, only affected the headers in the HTML files. The lower structures, e.g. paragraphs and reference components, did not change. Therefore, the code for converting SGML paragraphs to HTML paragraphs did not need modification. Nevertheless, this was the most difficult portion of the software project, taking more than 2 person-months of time to properly design, code, and validate the conversion.

Update Summary

Once the above technical problems were resolved, we focused on a much more difficult problem: the absence of summary information when a file was updated. The RCS archive software directly supports the entry of summary, therefore no development effort was needed. However, it was left to the discretion of the editorial staff to enter the appropriate summary. Consequently, many updates to the database do not have the accompanying summary.

In retrospect, it may not have been appropriate for the editorial staff to review the changes and enter a summary, because summary construction requires expert domain knowledge and semantic understanding of the changes. Therefore, without a policy decision in the editorial process, there was no guidance for the editors to summarize the changes they were making to the database. As an experiment, we provided explicit instructions on how to phrase the summary to some of the newly hired editorial staff. Their summaries were then evaluated before being considered for incorporation into the updates. Approximately 90% of the summaries were appropriate and were subsequently incorporated.

Currently, when reviewing the HTML format of the "rlog" output, the amount of summary information is variable. Without consistent summary information to provide key words and phrases, it was difficult for our WWW users to focus on a particular version. To find a particular change, a user has to step through each version sequentially.

Development Cost and Usage Patterns

The word-per-line implementation was instituted at the very beginning of conversion to SGML files because of the need for a word-level diff program for the editorial staff. All programming for historical access was coded using the "perl" language and UNIX shell. The SGML diff processing took 1 person-month and 500 lines of code; HTML formatting of versioned MIM entries took 2 person-months and 900 lines; and HTML conversions of "rlog" output and other support utilities took another 1 person-month and 500 lines. The historical retrievals are accomplished in real-time, i.e., the diff is executed, processed, and converted to HTML after the selection is made. Our experiments have shown that the software generates approximately 100 Kbytes of finished HTML text per minute on our production WWW server.

The extension for historical information was installed on our WWW server without public announcement in September 1994. Given the point-and-click nature of WWW browsers, we expected a few random users exploring this feature, but our WWW statistics show that more than a few were using this feature consistently. As of August 1995, usage is averaging 60 historical queries per week from outside users, of which 10 are from repeat users.

MIM editors have found the historical access very useful to track down quality control issues. For example, if a question arises about when a particular edit was made, our editors can use this feature to track the actual change event and its related source materials. In addition, our editors have found historical access useful to view all the updates to an entry since the time they last reviewed it, instead of comparing two printed copies side-by-side. Furthermore, this feature has allowed us to identify common editorial mistakes and apply appropriate remedies.

CONCLUSION

The addition of historical access to the MIM database was successful as a software development project. The amount of code and effort was reasonable for the complexity of the problem. The technical issues of word-level granularity, SGML tagging, and schema evolution were successfully resolved. However, as an informatics project, it was limited by several factors.

The first limiting factor is that the queries cannot be made against the state of the database at a given time in the past. As a compromise, all queries are performed on the current database; then the user can traverse back into the archives. We hope to develop a new search engine that can directly access the RCS archive to extract the past and present keywords.

The second limiting factor is that the log of versions for historical access has incomplete update summaries. To address this problem would require a policy decision by the editorial staff to support the proper entry of summary information. Afterwards, we can expand the training for summary construction to the remaining editorial staff.

The third limiting factor is that the current usage of this feature is from unguided exploration. We plan to make a general announcement to notify our users and to provide proper documentation and training for them in the near future.

We hope the above limiting factors will be resolved and a useful enhancement to the MIM database will be created. In addition, we hope the techniques developed and the issues learned from this project can be applied to other textual databases.

REFERENCES

1. Snodgrass, R.: Section Overview: Temporal Database Infrastructure. ACM Sigmod Record 23(1): 34-86, 1994.
2. Illustra User's Guide. Time Travel and Archiving. Release 2.1. Illustra Information Technologies, Inc. 1994.
3. Cellary, W., Jomier, G.: Consistency of Versions in Databases. In Bancilhon, F., Delobel, C., Kanellakis, P. (Eds): Building an Object-Oriented Database System. Morgan-Kaufman. p. 447-462, 1992.
4. Ramirez, J., Smith, L. Patterson, L.: Medical Information Systems: Characterization and Challenges. ACM Sigmod Record 23(3): 44-53, 1994.
5. McKusick, V. A.: Mendelian Inheritance in Man. 11th Edition. Johns Hopkins Press: Baltimore. 1994.
6. Harman, D., Benson, D., Fitzpatrick, L., Huntzinger, R., Goldstein, C.: IRX: An Information Retrieval System for Experimentation and User Applications. Proc. RIAO 88 Conf., 840-848, 1988.
7. Goldfarb, C. F.: The SGML Handbook. Oxford University Press: New York. 1992.
8. Li, P., Kramer, L., Pineo, S., Kulp, D.: Evolving a Legacy System: Restructing the Mendelian Inheritance in Man Database. 1994 SCAMC Proceedings. p. 344-348, 1994.
9. Tichy, W. F.: RCS-A system for Version Control. IEEE Software Practice and Experience. 15(7): 637-654, 1985.
10. Wall, L., Schwartz, R. L.: Programming perl. O'Reilly & Assoc.: Sebastopol. 1991.
11. Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H., Secret, A.: The World-Wide Web. Communications of ACM. 37(8): 76-82, 1994.
12. MacKenzie, D., Eggert, P., Stallman, R.: Comparing and Merging Files. Free Software Foundation, GNU documentation. 1993.

ACKNOWLEDGEMENTS

The authors wish to thank Drs. V. A. McKusick, D. Kingsbury, P. L. Pearson, K. Fasman, and Mr. C. Brunn for their support on this project. The work was funded by DOE DE-FC02-91ER61230.